EVA HACKATHON
A C A D E M Y

This is EVA Hackathon Academy assessment test, please try to solve as much problems as you can.

This assessment should take **48-hours to complete and submit**.

You may use any programming language of the following (C, C++, C#, Java, Kotlin, Objective-C, Swift, Python, PHP)

Submit instructions:

1. Each problem should be solved in a separate folder/project structure
2. Add your solution folder/project structure to a .rar file
3. Add all the solutions .rar files into one folder
4. Add the folder into a .rar file so that all your solutions would be in that .rar file

We will be waiting to hear from you, and we wish you the best of luck.

Problem 1

Given a string `s` representing a valid expression, implement a basic calculator to evaluate it, and return *the result of the evaluation*.

Note: You are **not** allowed to use any built-in function which evaluates strings as mathematical expressions, such as `eval()`.

Example 1:

```
Input: s = "1 + 1"

Output: 2
```

Example 2:

```
Input: s = " 2-1 + 2 "

Output: 3
```

Example 3:

```
Input: s = "(1+(4+5+2)-3)+(6+8)"

Output: 23
```

Constraints:

- `1 <= s.length <= 3 * 10^5`
- `s` consists of digits, `'+'`, `'-'`, `'('`, `')'`, and `' '`.
- `s` represents a valid expression.
- `'+'` is not used as a unary operation.
- `'-'` could be used as a unary operation but it has to be inside parentheses.
- There will be no two consecutive operators in the input.
- Every number and running calculation will fit in a signed 32-bit integer.

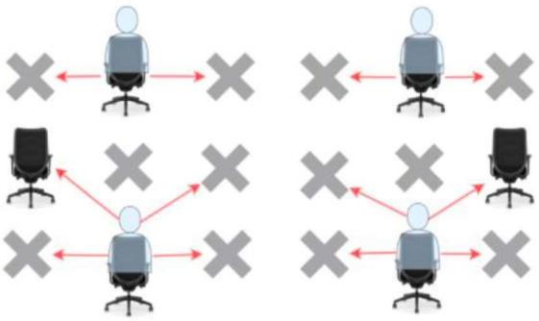
Problem 2

Given a m * n matrix seats that represent seats distributions in a classroom. If a seat is broken, it is denoted by '#' character otherwise it is denoted by a '.' character.

Students can see the answers of those sitting next to the left, right, upper left and upper right, but he cannot see the answers of the student sitting directly in front or behind him. Return the **maximum** number of students that can take the exam together without any cheating being possible..

Students must be placed in seats in good condition.

Example 1:



```
Input: seats = [["#",".","#","#",".","#"],
                [".","#","#","#","#","."],
                ["#",".","#","#",".","#"]]
Output: 4
Explanation: Teacher can place 4 students in available seats so they don't cheat on the
exam.
```

Example 2:

```
Input: seats = [[".","#"],
                ["#","#"],
                ["#","."],
                ["#","#"],
                [".","#"]]
Output: 3
Explanation: Place all students in available seats.
```

Example 3:

```
Input: seats = [["#",".",".",".","#"],
                [".","#",".","#","."],
                [".",".","#",".","."],
                [".","#",".","#","."],
                ["#",".",".",".","#"]]
Output: 10
Explanation: Place students in available seats in column 1, 3 and 5.
```

Constraints:

- `seats` contains only characters `'.'` and `'#'`.
- `m == seats.length`
- `n == seats[i].length`
- `1 <= m <= 8`
- `1 <= n <= 8`

Problem 3

Given an input string (`s`) and a pattern (`p`), implement wildcard pattern matching with support for `'?'` and `'*'` where:

- `'?'` Matches any single character.
- `'*'` Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

Example 1:

```
Input: s = "aa", p = "a"

Output: false

Explanation: "a" does not match the entire string "aa".
```

Example 2:

```
Input: s = "aa", p = "*"

Output: true

Explanation: '*' matches any sequence.
```

Example 3:

```
Input: s = "cb", p = "?a"

Output: false

Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.
```

Example 4:

```
Input: s = "adceb", p = "*a*b"

Output: true

Explanation: The first '*' matches the empty sequence, while the second '*' matches the
substring "dce".
```

Example 5:

```
Input: s = "acdcb", p = "a*c?b"

Output: false
```

Constraints:

- `0 <= s.length, p.length <= 2000`
- `s` contains only lowercase English letters.
- `p` contains only lowercase English letters, `'?'` or `'*'`.

Problem 4

Given n, calculate the sum LCM(1,n) + LCM(2,n) + .. + LCM(n,n), where LCM(i,n) denotes the Least Common Multiple of the integers i and n.

Input

The first line contains T the number of test cases. Each of the next T lines contain an integer n.

Output

Output T lines, one for each test case, containing the required sum.

Example

```
Sample Input:
3
1
2
5

Sample Output:
1
4
55
```

Constraints

1 <= T <= 300000
1 <= n <= 1000000

Problem 5

For the given number *n* find the minimal positive integer divisable by *n*, with the sum of digits equal to *n*.

Input

t – the number of test cases, then *t* test cases follow. (t <= 50)
Test case description:
n - integer such that 0 < *n* <= 1000

Output

For each test case output the required number (without leading zeros).

Example

```
Input:
2
1
10

Output:
1
190
```

Problem 6

The best pairing in this example is created by linking the first and second offices together, and linking the third and fourth offices together. This uses k = 2 cables as required, where the first cable has length 3km - 1km = 2 km, and the second cable has length 6km - 4km = 2 km. This pairing requires a total of 4km of network cables, which is the smallest total possible.

Input

Multiple test cases, the number of them will be given at the very first line.

For each test case:

The first line of input will contain the integers n and k, representing the number of offices on the street (2 <= n <= 100 000) and the number of available network cables (1 <= k <= n/2).

The following n lines will each contain a single integer (0 <= s <= 1 000 000 000), representing the distance of each office from the beginning of the street. These integers will appear in sorted order from smallest to largest. No two offices will share the same location.

Output

Output should consist of a single positive integer, giving the smallest total length of network cable required to join 2k distinct offices into k pairs.

Example

Input:

```
1
5 2
1
3
4
6
12
```

Output:

```
4
```

Explanation

```
The sample input above represents the example scenario described earlier.
```

You run an IT company that backs up computer data for large offices. Backing up data is not fun, and so you design your system so that the different offices can back up each others' data while you sit at home and play computer games instead.

The offices are all situated along a single street. You decide to pair up the offices, and for each pair of offices you run a network cable between the two buildings so that they can back up each others' data.

However, network cables are expensive. Your local telecommunications company will only give you k network cables, which means you can only arrange backups for k pairs of offices (2k offices in total). No office may belong to more than one pair (that is, these 2k offices must all be different). Furthermore, the telecommunications company charges by the kilometre. This means that you need to choose these k pairs of offices so that you use as little cable as possible. In other words, you need to choose the pairs so that, when the distances between the two offices in each pair are added together, the total distance is as small as possible.

As an example, suppose you had five clients with offices on a street as illustrated below. These offices are situated 1 km, 3 km, 4 km, 6km and 12km from the beginning of the street. The telecommunications company will only provide you with k = 2 cables.